SpringBoot:

****

Sprint Boot is a project developed on top of Spring Frame Work. Its provides an easier and faster way to setup, Configure and run both simple and web based applications.

Spring Framework + Embedded HTTP Servers (Tomcat, Jetty) + XMl based Configuration/@Configuration = Spring Boot.

In short, Spring Boot is the combination of Spring Framework and Embedded Servers.

It is developed by Pivotal Team. In October 2012, Mike Youngstrom created a feature request in spring jira asking for support for containerless web application architectures in spring framework. He talked about configuring web container services within a spring container bootstrapped from the main method! Here is an excerpt from the jira request,

This request lead to the development of spring boot project starting sometime in early 2013. In April 2014, spring boot 1.0.0 was released. Current version of SpringBoot is 2.4.3.

Why we should use the SpringBoot

==========

1. Less configurations.
2. Speed development.
3. Easy to implement the production ready features like metrics, health checks.
4. Easy integration with any module like JPA.
5. Embedded Servers.
6. YAML Support

Limitations of Spring Boot

==========

Spring Boot can use dependencies that are not going to be used in the application. These dependencies increase the size of the application.

Ways to create the SpringBoot Application:

==========

1. By using STS, Select Spring Starter Project.

2. By Using Spring initializer https://start.spring.io

3. Command Line Interface (CLI)

---

1. Download the CLI zip

2. Setup the extracted CLI folder path in windows environment variables path.

3. Run this command: spring run

Software versions required for Creating SpringBoot Projects:

==========

Spring boot 2.4.4 required

Java 8 version.

Gradle 6.3 +

Tomcat 9

Jetty 2.4.4

Minimum required version to run the springboot app is Java 7.

Whats new in SpringBoot 2.x

==========

1. Java 8 baseline and Java 9 support.

2. JUnit 5's vintage engine is also included by default that supports existing JUnit 4-based test classes. We can also use JUnit 4 and JUnit 5

3. Spring 5 support

4. Classpath scanning support for @ConfigurationProperties

5. Supports Hibernate 5.2

SpringBoot Annotations:

==========

@SpringBootApplication:

==========

We use this annotation to mark the main class of a Spring Boot application. When we run the application its enable the auto configuration.

```
@SpringBootApplication
class VehicleFactoryApplication {

   public static void main(String[] args) {
      SpringApplication.run(VehicleFactoryApplication.class, args);
   }
}
```

This annotations combination of below

1. @EnableAutoConfiguration
2. @Configuration
3. @ComponentScan

@EnableAutoConfiguration:

==========

This will enable the auto configuration. It will configure the all the classes available in class path, It means while creating Spring Boot app if wee add the we dependency it will configure all dependencies related to it.

We will use the @EnableAutoConfiguration along with @Configuration

```
@Configuration
```

```
@EnableAutoConfiguration
class VehicleFactoryConfig {}
```

@Configuration:

==========

Its a class level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions and tag these beans to application context.

@ComponentScan:

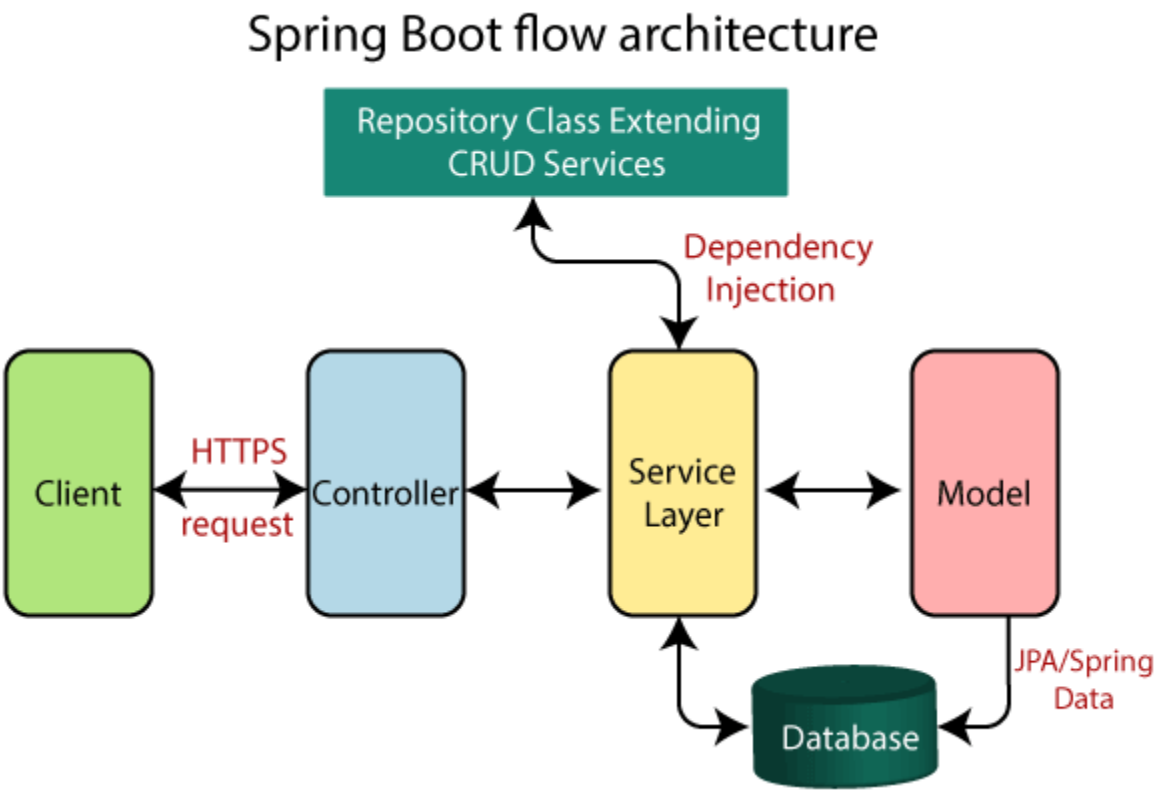
==========

Its responsible for scanning the base and sub packages. The @ComponentScan annotation is used with the @Configuration annotation.The basePackageClasses attribute is a type-safe alternative to basePackages. When you specify basePackageClasses, Spring will scan the package (and subpackages) of the classes you specify.

```
@Configuration
@ComponentScan(basePackages = {
      "guru.springframework.blog.componentscan.example.demopackageA",
      "guru.springframework.blog.componentscan.example.demopackageD",
      "guru.springframework.blog.componentscan.example.demopackageE"
   },
   basePackageClasses = DemoBeanB1.class)
```

//

SpringBoot Architecture:



Spring Boot is a module of the Spring Framework. It is used to create stand-alone, production-ready Spring Based Applications with minimum efforts. It is developed on top of the core Spring Framework.

Before understanding the Spring Boot Architecture, we must know the different layers and classes present in it. There are four layers in Spring Boot are as follows:

Presentation Layer:

****

The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of views i.e., frontend part.

Business Layer:

****

The business layer handles all the business logic. It consists of service classes and uses services provided by data access layers. It also performs authorization and validation.

Persistence Layer:

****

The persistence layer contains all the storage logic and translates business objects from and to database rows.

Database Layer:

****

In the database layer, CRUD (create, retrieve, update, delete) operations are performed.

Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, except one thing: there is no need for DAO and DAOImpl classes in Spring boot.

1.The client makes the HTTP requests (PUT or GET).

2.The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.

3.In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.

4.A JSP page is returned to the user if no error occurred.

//

Spring Initializr:

==========

Spring Initializr is a web-based tool provided by the Pivotal Web Service. With the help of Spring Initializr, we can easily generate the structure of the Spring Boot Project.

It also provides various options for the project that are expressed in a metadata model. The metadata model allows us to configure the list of dependencies supported by JVM and platform versions, etc. It serves its metadata in a well-known that provides necessary assistance to third-party clients.

//
////////////////////////////////////

Spring Boot Dependency Management:

==========

Spring Boot manages dependencies and configuration automatically. Each release of Spring Boot provides a list of dependencies that it supports. The list of dependencies is available as a part of the Bills of Materials (spring-boot-dependencies) that can be used with Maven. So, we need not to specify the version of the dependencies in our configuration. Spring Boot manages itself. Spring Boot upgrades all dependencies automatically in a consistent way when we update the Spring Boot version.

Advantages of Dependency Management:

==========

1.It provides the centralization of dependency information by specifying the Spring Boot version in one place. It helps when we switch from one version to another.

2.It avoids mismatch of different versions of Spring Boot libraries.

3.We only need to write a library name with specifying the version. It is helpful in multi-module projects.

//
////////////////////////////////////

SpringBoot Starter:

==========

Spring Boot provides a number of starters that allow us to add jars in the classpath. Spring Boot built-in starters make development easier and rapid. Spring Boot Starters are the dependency descriptors.

In the Spring Boot Framework, all the starters follow a similar naming pattern: spring-boot-starter-*, where * denotes a particular type of application. For example, if we want to use Spring and JPA for database access, we need to include the spring-boot-starter-data-jpa dependency in our pom.xml file of the project.

Example:

==========

spring-boot-starter-web

spring-boot-starter-security

spring-boot-starter-mail

spring-boot-starter-jpa

Spring Boot Production Starters:

---

spring-boot-starter-actuator

Spring Boot Technical Starters:

---

spring-boot-starter-tomcat

spring-boot-starter-jetty

spring-boot-starter-logging

Third-Party Starters:

==========

We can also include third party starters in our project. But we do not use spring-boot-starter for including third party dependency. The spring-boot-starter is reserved for official Spring Boot artifacts. The third-party starter starts with the name of the project. For example, the third-party project name is abc, then the dependency name will be abc-spring-boot-starter.

Spring Boot Starter Parent:

==========

The spring-boot-starter-parent is a project starter. It provides default configurations for our applications. It is used internally by all dependencies. All Spring Boot projects use spring-boot-starter-parent as a parent in pom.xml file.

The spring-boot-starter-parent inherits dependency management from spring-boot-dependencies. We only need to specify the Spring Boot version number.

Spring Boot Starter Web:

==========

There are two important features of spring-boot-starter-web:

It is compatible for web development

Auto configuration

Spring web uses Spring MVC, REST and Tomcat as a default embedded server. The single spring-boot-starter-web dependency transitively pulls in all dependencies related to web development. It also reduces the build dependency count.

Spring Boot Embedded Web Server:

==========

Each Spring Boot application includes an embedded server. Embedded server is embedded as a part of deployable application. The advantage of embedded server is, we do not require pre-installed server in the environment. With Spring Boot, default embedded server is Tomcat. Spring Boot also supports another two embedded servers:

Jetty Server

Undertow Server

How to use the different embedde servers?

==========

If we want to use the Tomcat we just need to exclude the Jetty manually in pom.xml/build.gradle.

Spring Boot Starter Actuator:

==========

Spring Boot Actuator is a sub-project of the Spring Boot Framework. It includes a number of additional features that help us to monitor and manage the Spring Boot application.If we want to get production-ready features in an application, we should use the Spring Boot actuator.

This starter will provide the folowing features:

----

1. Health Check

2. Metrics

3. Audit

Spring Boot actuator properties:

==========

Spring Boot enables security for all actuator endpoints. It uses form-based authentication that provides user Id as the user and a randomly generated password. We can also access actuator-restricted endpoints by customizing basicauth security to the endpoints. We need to override this configuration by management.security.roles property

management.security.enabled=true

management.security.roles=ADMIN

security.basic.enabled=true

security.user.name=admin

security.user.passowrd=admin

//
//

Spring Boot Auto-configuration:

==========

We can enable the auto-configuration feature by using the annotation @EnableAutoConfiguration. But this annotation does not use because it is wrapped inside the @SpringBootApplication annotation. The annotation @SpringBootApplication is the combination of three annotations: @ComponentScan, @EnableAutoConfiguration, and @Configuration. However, we use @SpringBootApplication annotation instead of using @EnableAutoConfiguration.

Disable Auto-configuration Classes:

==========

We can also disable the specific auto-configuration classes, if we do not want to be applied. We use the exclude attribute of the annotation @EnableAutoConfiguration to disable the auto-configuration classes

```
@Configuration(proxyBeanMethods = false)
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration
{
}
```